

1. Les attaques MITM : HTTPS
2. Le phishing : Eduquer les visiteurs
3. Les attaques CSRF : Token CSRF
4. Les modifications du DOM : Faire les vérifications au backend
5. Les injections : Figer la forme de la requete SQL et désactiver les balises HTML
6. Le brute force : Mettre une limite sur le nombre d'essais



# Que sont les injections ?



# PREVENTION

Il faut désactiver le code SQL ou HTML dans les données que l'on reçoit au niveau du payload.

Si nous ne les désactivons pas, on s'expose au risque que les codes injectés peuvent s'exécuter.

1. Injections SQL : pour désactiver le code SQL, il existe 2 manières : soit on utilise les requêtes préparées, soit on utilise la fonction quote pour désactiver les codes SQL directement
2. Injections XSS : désactiver le code HTML dans les variables - utiliser la fonction htmlentities pour désactiver les codes HTML directement

# Désactivation du code SQL en entrée

# Rappel

```
function check()
{
    // On récupère le username et le password du payload
    $username = $_POST['username'];
    $password = $_POST['password'];

    // On crée une connexion vers la base de données (comme mysqli)
    $connexion = new PDO('mysql:host=localhost;dbname=patissor', 'root', '');

    // On exécute la requête SQL pour chercher l'utilisateur avec ce $username et $password
    $requete = "SELECT COUNT(id) AS nbr FROM user WHERE username = '$username' AND password = '$password'";
    $resultat = $connexion->query($requete);

    // On récupère le résultat de la requête SQL dans $row
    $row = $resultat->fetch();

    // On vérifie si nbr > 0 (donc si un utilisateur a été trouvé en base)
    if ($row['nbr'] > 0)
    {
        // On redirige vers l'espace client
        header('LOCATION: /espace-client');
        die();
    }
}
```

Méthode 1 : Utiliser les requêtes préparées  
pour figer la forme de la requête



## Avant le relooking :

```
// On exécute la requete SQL pour chercher l'utilisateur avec ce $username et $password
$requete = "SELECT COUNT(id) AS nbr FROM user WHERE username = '$username' AND password = '$password'";
$resultat = $connexion->query($requete);
```

## Après le relooking :

```
// 1. On écrit la structure de la requete avec des trous "?"
$requete = "SELECT COUNT(id) AS nbr FROM user WHERE username = ? AND password = ?";
// 2. On la fige
$resultat = $connexion->prepare($requete);
// 3. On execute la requete avec $username et $password (désactivation automatique du code)
$resultat->execute([$username, $password]);
```

Méthode 2 : Utiliser une fonction pour désactiver le code SQL directement : la fonction `quote`

Il existe une autre fonction `mysqli_real_escape_string` mais elle est obsolète

## Avant le relooking :

```
// On récupère le username et le password du payload
$username = $_POST['username'];
$password = $_POST['password'];


// On crée une connexion vers la base de données (comme mysqli)
$connexion = new PDO('mysql:host=localhost;dbname=patissor', 'root', '');
```

## Après le relooking (avec la méthode quote) :

```
// On récupère le username et le password du payload
$username = $_POST['username'];
$password = $_POST['password'];

// On crée une connexion vers la base de données (comme mysqli)
$connexion = new PDO('mysql:host=localhost;dbname=patissor', 'root', '');

// Dans $username, on désactive le code SQL, puis on le remet dedans
$username = $connexion->quote($username);
// Dans $password, on désactive le code SQL, puis on le remet dedans
$password = $connexion->quote($password);
```



Méthode 1 ou méthode 2 : vous connaissez désormais les 2 méthodes pour se prémunir d'une injection SQL.

A votre avis, est-ce la méthode 1  
ou la méthode 2 la meilleure ?



Méthode 1 ou méthode 2 : vous connaissez désormais les 2 méthodes pour se prémunir d'une injection SQL. Ce qui est recommandé, c'est d'utiliser **les requêtes préparées** qui sont beaucoup plus rapides.

# Désactivation du code HTML en entrée

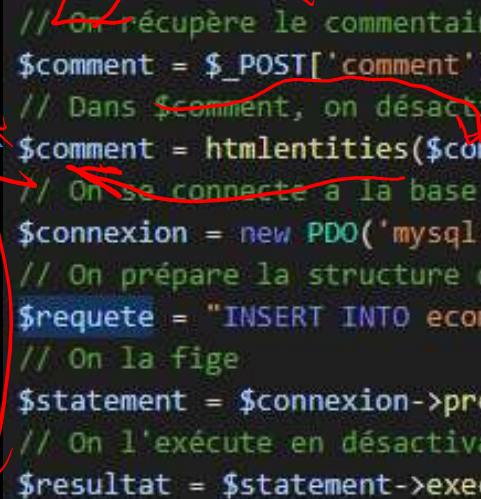
Utiliser **la fonction htmlentities** pour  
désactiver le code HTML directement



Avant le  
relooking :

```
// On récupère le commentaire du payload
$comment = $_POST['comment'];
// On se connecte a la base de données
$connexion = new PDO('mysql:host=localhost;dbname=ecom', 'root', '');
// On prépare la structure de la requete
$requete = "INSERT INTO ecom_comment(content) VALUES(?)";
// On la fige
$stmtement = $connexion->prepare($requete);
// On l'exécute en désactivant le code SQL dans $comment
$resultat = $statement->execute([$comment]);
```

Après le  
relooking :



```
// On récupère le commentaire du payload
$comment = $_POST['comment'];
// Dans $comment, on désactive le code HTML et on le remet
$comment = htmlentities($comment);
// On se connecte a la base de données
$connexion = new PDO('mysql:host=localhost;dbname=ecom', 'root', '');
// On prépare la structure de la requete
$requete = "INSERT INTO ecom_comment(content) VALUES(?)";
// On la fige
$stmtement = $connexion->prepare($requete);
// On l'exécute en désactivant le code SQL dans $comment
$resultat = $statement->execute([$comment]);
```

Par ailleurs, dans le dernier exemple, nous avons un **code complètement protégé des injections**, à la fois contre les injections SQL (car il utilise les **requêtes préparées**) et contre les injections XSS (car il utilise la fonction **htmlentities**)

CERTIF  
ACADEMY